Core Architecture Optimization for Heterogeneous Chip Multiprocessors

Rakesh Kumar Dean M. Tullsen
Dept of Computer Science and Engineering
University of California, San Diego
La Jolla, CA 92093-0404

Norman P. Jouppi
HP Labs
1501 Page Mill Road
Palo Alto, CA 94304

ABSTRACT

Previous studies have demonstrated the advantages of single-ISA heterogeneous multi-core architectures for power and performance. However, none of those studies examined how to design such a processor; instead, they started with an assumed combination of pre-existing cores.

This work assumes the flexibility to design a multi-core architecture from the ground up and seeks to address the following question: what should be the characteristics of the cores for a heterogeneous multi-processor for the highest area or power efficiency? The study is done for varying degrees of thread-level parallelism and for different area and power budgets.

The most efficient chip multiprocessors are shown to be heterogeneous, with each core customized to a different subset of application characteristics – no single core is necessarily well suited to all applications. The performance ordering of cores on such processors is different for different applications; there is only a partial ordering among cores in terms of resources and complexity. This methodology produces performance gains as high as 40%. The performance improvements come with the added cost of customization.

Categories and Subject Descriptors: C.1.2 [Processor Architectures]: Multiprocessors

General Terms: Design, Performance

Keywords: heterogeneous chip multiprocessors, computer architecture, multi-core architectures

1. INTRODUCTION

Multiple-core processor architectures are becoming increasingly attractive as an option to provide high instruction throughput while keeping power and complexity under control. But multi-core processors also give the designer more flexibility to meet specific performance/power goals. In particular, single-ISA heterogeneous multi-core architectures have been shown to provide significant power and performance advantages for chip multiprocessors (CMPs) [10, 11]. Such an architecture consists of multiple core types on the same die; each core representing a different point in the power-performance continuum. Applications are mapped to cores in such a way that each application executes on a core that best fits its runtime resource requirements. This results in higher overall computational efficiency than conventional homogeneous CMPs.

While the previous proposals demonstrated the benefits of heterogeneity, they gave no insight into what constitutes, or how to arrive at, a good heterogeneous design. Previous work assumed a given heterogeneous architecture. More specifically, those architectures were composed of existing architectures, either different generations of the same processor family [11, 10, 4, 6], or voltage and frequency scaled editions of a single processor [2, 3, 5, 9]. While these architectures surpassed similar homogeneous designs, they failed to reach the full potential of heterogeneity, for three reasons. First, the use of pre-existing designs presents low flexibility in choice of cores. Second, those core choices maintain a monotonic relationship, both in design and performance – for example, the most powerful core is bigger or more complex in every dimension and the performance-ordering of the cores is the same for every application. Third, all cores considered perform well for a wide variety of applications — we show that the best heterogeneous designs are composed of specialized core architectures.

A heterogeneous architecture, and particularly a fully custom heterogeneous processor not necessarily composed of pre-existing cores, incurs additional costs in design, verification, and testing. A key goal of this research is to evaluate the full benefits of these architectures, so that this trade-off can be more appropriately evaluated by processor manufacturers.

In actually deriving the best designs for a variety of multiprogramming workloads, power and area constraints, level of threading, etc., we make three significant contributions. First, we re-evaluate the benefits of heterogeneity in power and area efficient architectures, showing new benefits and higher gains. Performance improvements of up to 40% are shown. Second, we demonstrate methodologies for arriving at good heterogeneous designs – we examine both those that find the best designs but do not scale well to larger design spaces, and those that scale yet still find good architectures. Third, by actually finding the best designs across many different assumptions and constraints, we identify a number of key principles critical to the effective design of future chip multiprocessors.

More specifically, this study leads to several conclusions regarding effective heterogeneous CMP design.

- The most efficient heterogeneous multiprocessor is not constructed of cores that make good general-purpose uniprocessor cores, or even those cores that would appear in a good homogeneous multiprocessor architecture.

PACT'06, September 16–20, 2006, Seattle, Washington, USA.

- The best way to design a heterogeneous CMP is by tuning each individual core for a class of applications with common characteristics.
- Customizing cores to subsets of workloads results in processors that are typically non-monotonic (i.e., there is no strict gradation among cores in terms of overall performance or complexity).
- Performance advantages of heterogeneous, and even non-monotonic, multiprocessors continue to hold even for a collection of completely homogeneous workloads. In those cases, such processors exploit the diversity across different workloads.

The rest of the paper is organized as follows. Section 2 describes prior related work. Section 3 describes the approach followed to navigate the design space and arrive at the best designs for a given set of workloads. Section 4 discusses the benefits of customization. Section 5 gives the methodology followed for our evaluations and describes the area, power, and performance models. Section 6 provides the results of our experiments. Section 7 concludes.

2. RELATED WORK

Prior work on single-ISA heterogeneous multi-core architectures demonstrates the benefits of heterogeneity for both power/performance efficiency and area/performance efficiency. However, those studies focus on the benefits of an assumed design, and thus give little insight into what constitutes, or how to arrive at, a good heterogeneous design.

Initial proposals for heterogeneous multi-core architectures demonstrated the power efficiency of such architectures. Kumar, *et al.* [10] propose single-ISA heterogeneous multi-core architectures for processor power reduction. The proposal consists of cores from different generations of the Alpha processor family on the same die. They consider a single application running at a time that gets mapped intelligently to the right core. The energy benefits of heterogeneous multi-core architectures is also explored by Ghiasi and Grunwald [4]. They consider single-ISA, heterogeneous cores of different frequencies belonging to the x86 family, and use them to control the thermal characteristics of a system. Applications run simultaneously on multiple cores and the operating system monitors and directs applications to the appropriate job queues. Grochowsky, *et al.* [6] compare voltage/frequency scaling, asymmetric (heterogeneous) cores, variable-sized cores, and speculation as means to reduce the energy per instruction and find that heterogeneous cores result in the most benefit.

Another set of proposals use heterogeneous multi-core architectures to improve processor performance for fixed area and power budgets. Kumar, *et al.* [11] demonstrate the performance advantages of heterogeneous multi-core architectures for multi-programmed heterogeneous workloads. They consider multiprocessors consisting of EV5 and/or EV6 cores and show that on-chip heterogeneity results in more efficient computation and helps target a broad spectrum of thread-level parallelism. Morad, *et al.* [15] explore the theoretical advantages of placing asymmetric core clusters in multiprocessor chips. They show that asymmetric core clusters are expected to achieve higher performance per area and higher performance for a given power envelope. The analysis is extended in [16]. Annavaram, *et al.* [2] evaluate the benefits of heterogeneous multiprocessing to minimize the execution times of multi-threaded programs containing nontrivial parallel and sequential phases, while keeping the CMP's total power consumption within a fixed budget. They report significant speedups.

Balakrishanan, *et al.* [3] seek to understand the impact of such an architecture on software. They show, using a hardware prototype, that asymmetry can have significant impact on the performance of a wide range of commercial applications.

The power-performance trade-offs for multi-core architectures was also studied recently by Li, *et al.* [13]. That work does not consider heterogeneous chip multiprocessors, however.

3. FROM WORKLOADS TO A MULTI-CORE DESIGN

The goal of this research is to identify the characteristics of cores that combine to form the best heterogeneous architectures, and also demonstrate principles for designing such an architecture. Such a methodology would start with a set of applications and a set of constraints on the processor. It should then identify the best architecture for that workload, given some objective function to evaluate the goodness of an architecture.

Because this methodology requires that we accurately reflect the wide diversity of applications (their parallelism, their memory behavior), running on widely varying architectural parameters, there is no real shortcut to using simulation to characterize these combinations.

The design space for even a single processor is large, given the flexibility to change various architectural parameters; however, the design space explodes when considering the combined performance of multiple different cores on arbitrary permutations of the applications. Hence, we make some simplifying assumptions that make this problem tractable so that we navigate through the search space faster; however, we show that the resulting methodology still results in the discovery of very effective multi-core design points.

First, we assume that the performance of individual cores is separable – that is, that the performance of a four-core design, running four applications, is the sum (or the sum divided by a constant factor) of the individual cores running those applications in isolation. This is an accurate assumption if the cores do not share L2 caches (which we validate in the Appendix) or memory channels.

However, we also show in the Appendix that this methodology still makes good design decisions with shared L2 caches for our workloads. This assumption dramatically accelerates the search because now the single-thread performance of each core (found using simulation) can be used to estimate the performance of the processor as a whole without the need to simulate all 4-thread permutations.

Since we are interested in the highest performance that a processor can offer, we assume good static scheduling of threads to cores. Thus, the performance of four particular threads on four particular cores is the performance of the best static mapping. However, this actually represents, in some sense, a *lower bound* on performance. Prior work has shown that the ability to migrate threads dynamically during execution only increases the benefits of heterogeneity [11] as it exploits intra-thread diversity – we show in Section 6 that it continues to hold true for the best heterogeneous designs that we come up with under the static scheduling assumption.

To further accelerate the search, we consider only major blocks to be configurable, and only consider discrete points. For example, we consider 2 instruction queue sizes (rather than all the intermediate values) and 4 cache configurations (per cache). But we consider only a single branch predictor, because the area/performance trade-offs of different sizes had little effect in our experiments. Values that are expected to be correlated (e.g., size of re-order buffer and number of physical registers) are scaled together instead of separately. This methodology might appear to be crude for an important

Issue width	1 , 2 , 4	Int-FP PhysReg-ROB (OOO)	64-64-32, 128-128-64
I-Cache	8KB DM, 16KB 2way, 32KB 4way, 64KB 4way	L2 Cache	1MB/core, 4-way, 12cycle access
D-Cache	8KB DM, 16KB 2way, 32KB 4way, 64KB 4way dual ported	Memory Channel	533MHz, doubly-pumped, RDRAM
FP-IntMul-ALU units.	1-1-2, 2-2-4	ITLB-DTLB	64, 28 entries
IntQ-fpQ (OOO)	32-16, 64-32	Ld/St Queue	32entries

Table 1: Various Parameters and their possible values for configuration of the cores.

commercial design, but we believe that even in that environment this methodology would find a design very much in the neighborhood of the best design. Then, a more careful analysis could be done of the immediate neighborhood, considering structure sizes at a finer granularity and considering particular choices for smaller blocks we did not vary.

We only consider and compare processors with a fixed number (4) of cores. It would be interesting to also relax that constraint in our designs, but we did not do so for the following reasons. Accurate comparisons would be more difficult, because the interconnect and cache costs would vary. Second, it is shown both in this work (Section 6) and in previous work [11] that heterogeneous designs are much more tolerant than homogeneous when running a different number of threads than the processor is optimized for. However, the methodology shown here need only be applied multiple times (once for each possible core count) to fully explore the larger design space, assuming that an accurate model of the off-core resources was available.

The above assumptions allow us to model performance for various combinations of cores for various permutations of our benchmarks, thus evaluating the expected performance of the possible homogeneous and heterogeneous processors for various area and power budgets.

To search through the design space for a given set of workloads we follow two techniques – exhaustive search and efficient search. Our algorithm for finding the best design typically is an exhaustive search of all core combinations, accounting for every permutation of our benchmarks on each combination. This approach ensures that we do indeed find the best combination in each case. While this approach works for our workloads and architectural variables, considering more benchmarks and more architectural options will quickly make the exhaustive approach impractical. In Section 6.5, we examine more efficient search algorithms and quantify how closely they come to identifying the best design.

4. CUSTOMIZING CORES TO WORKLOADS

One of the biggest advantages of creating a heterogeneous processor as a custom design is that the cores can be chosen in an unconstrained manner as long as the processor budgetary constraints are satisfied. We define *monotonicity* to be a property of a multi-core architecture where there is a total ordering among the cores in terms of performance and this ordering remains the same for all applications. For example, a multiprocessor consisting of EV5 and EV6 cores is a monotonic multiprocessor. This is because EV6 is strictly superior to EV5 in terms of hardware resources and virtually always performs better than EV5 for a given application given the same cycle time and latencies. Similarly, for a multi-core architecture with identical cores, if the voltage/frequency of a core is set lower than the voltage/frequency of some other core, it will always provide less performance, regardless of application. Fully customized monotonic designs represent the upper bound (albeit a high one) on the benefits possible through previously proposed heterogeneous architectures.

As we show in this paper, monotonic multiprocessors may not provide the "best fit" for various workloads and hence result in inefficient mapping of applications to cores. For example, in the results shown in [10], *mcf*, despite having very low ILP, consistently gets mapped to the EV6 or EV8- core for various energy-related objective functions, because of the larger caches on these cores. Yet it fails to take advantage of the complex execution capabilities of these cores, and thus still wastes energy unnecessarily.

Doing a custom design of a heterogeneous multi-core architecture allows us to relax the monotonicity constraint. That is, it is possible for a particular core of the multiprocessor to be the highest performing core for some application but not for others. For example, if one core is in-order, scalar, with 32KB caches, and another core is out-of-order, dual-issue, with larger caches, applications will always run best on the latter. However, if the scalar core had larger L1 caches, then it might perform better for applications with low ILP and large working sets, while the other would likely be best for jobs with high ILP and smaller working sets.

The advantage of non-monotonicity is that now different cores on the same die can be customized to different classes of applications, which was not the case with previously studied designs.

5. METHODOLOGY

This section discusses the various methodological challenges of this research, including modeling the power, real estate, and performance of the heterogeneous multi-core architectures.

5.1 Modeling of CPU Cores

For all our studies in this paper, we model 4-core multiprocessors assumed to be implemented in 0.10 micron, 1.2V technology. Each core on a multiprocessor, either homogeneous or heterogeneous, has a private L2 cache and each L2 bank has a corresponding memory controller. The ITRS roadmap [1] confirms that sufficient pins are available to support four memory controllers for the assumed technology. Assuming private L2 caches reduces the dimensions of the design; however, we also consider a shared L2 cache (of the same total size) in the Appendix.

We consider both in-order cores and out-of-order (OOO) cores for this study. We base our OOO processor microarchitecture model on the MIPS R10000, and our in-order cores on the Alpha EV5 (21164). We evaluate 480 cores as possible building blocks for constructing the multiprocessors. This represents all possible distinct cores that can be constructed by changing the parameters listed in Table 1. The various values that were considered are listed in the table as well. We assumed a gshare branch predictor [14] with 8k entries for all the cores. Out of these 480 cores, there are 96 distinct in-order cores and 384 distinct out-of-order cores. The number of distinct 4-core multiprocessors that can be constructed out of 480 distinct cores is over 2.2 billion.

Structure	Methodology	Assumptions
L1 caches	[20]	Parallel data/tag access
TLBs	[20],[7]	
RegFiles	[20],[17]	$2 \times IW$ RP, IW WP
Execution Units	[7]	
RenameTables	[20][17]	$3 \times IW$ RP, IW WP
ROBs	[20]	IW RP, IW WP, 20b-entry,6b-tag
IQs(CAM arrays)	[20]	IW RP, IW WP, 40b-entry,8b-tag
Ld/St Queues	[20]	64b-addressing,40b-data

Table 2: Area and power estimation methodology and relevant assumptions for various hardware structures. Renaming for OOO cores is assumed to be done using RAM tables. IW refers to issue-width, WP to a write-port, and RP to a read-port.

Other parameters that are kept fixed for all the cores are also listed in Table 1. The various miss penalties and L2 cache access latencies for the simulated cores were determined using CACTI [20].

All evaluations are done for multiprocessors satisfying a given aggregate area and power budget for the 4 cores. We do not expect the memory and interconnection subsystem to vary significantly with the core type for a given number of cores. We also confirmed that L2's contribution to overall power consumption did not vary significantly between four-core designs taking up the same area, even when the total number of serviced memory requests differed. Hence, we do not concern ourselves with the area and power consumption of anything other than the cores for this study.

5.2 Modeling Power and Area

In this paper, the area budget refers to the sum of the area of the 4 cores of a processor (the L1 cache being part of the core), and the power budget refers to the sum of the worst case power of the cores of a processor. Specifically, we consider peak activity power, as this is a critical constraint in the architecture and design phase of a processor. Static power is not considered explicitly in this paper (though it is typically proportional to area, which we do consider).

We model the peak activity power and area consumption of each of the key structures in a processor core using a variety of techniques. Table 2 lists the methodology and assumptions used for estimating area and power overheads for various structures. Table 3 shows the area and power values for various parameterized hardware structures that make up a core for different issue widths. Notice that some of the structures listed are for OOO cores only.

To get total area and power estimates, we assume that the area and power of a core can be approximated as the sum of its major pieces. In reality, we expect that the unaccounted-for overheads will scale our estimates by constant factors (leakage power scaling might not be linear). In that case, all our results will still be valid.

Figure 1 shows the area and power of the 480 cores used for this study. As can be seen, the cores represent a significant range in terms of power (4.1-16.3W) as well as area (3.3-22mm^2). For this study, we consider 4-core multiprocessors with different area and peak power budgets. There is a significant range in the area and power budget of the 4-core multiprocessors that can be constructed out of these cores. Area can range from 13.2mm^2 to 88mm^2. Power can range from 16.4W to 65.2W.

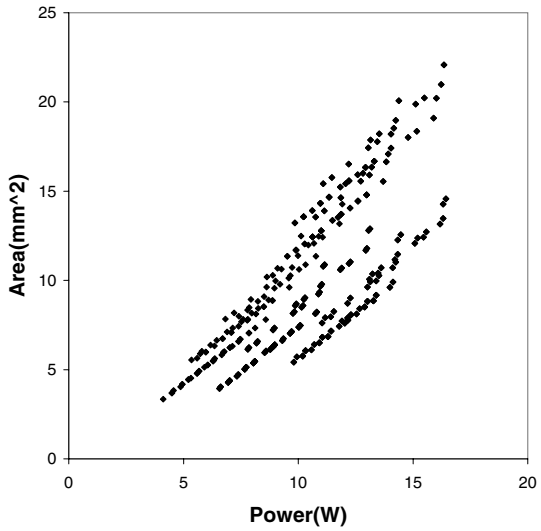

Figure 1: Area and Power of the cores

5.3 Modeling Performance

This section describes the workloads used for evaluation, the performance evaluation methodology, and the evaluation metric.

5.3.1 Workloads

All our evaluations are done for multiprogrammed workloads. Table 4 lists the ten benchmarks used for constructing workloads. Seven benchmarks are from the SPEC suite. These benchmarks are chosen in the following way. We simulated *all* 26 benchmarks from the SPEC suite for 250 million cycles using the EV5 processor model after fast-forwarding for an appropriate number of instructions [19]. Then benchmarks were classified into *processor bound* or *bandwidth bound* based on the number of main memory references per instruction. Seven benchmarks were then chosen from these two sets in proportion to the occurrence of these classes of benchmarks in the SPEC suite. Hence, the chosen SPEC benchmarks are intended to represent the *entire* SPEC suite. We also chose *groff*, *deltablue* and *adpcmc* from the IBS, OOCSB and Mediabench suites respectively. Choosing these three additional benchmarks recognizes the existence of other kinds of application behavior that are not displayed by the SPEC benchmarks, while still considering SPEC representative of a wide variety of applications.

Every multiprocessor is evaluated on two classes of workloads. The *all different* class consists of all possible 4-threaded combinations that can be constructed such that each of the 4 threads running at a time is different. The *all same* consists of all possible 4-threaded combinations that can be constructed such that all the 4 threads running at a time are the same. For example, *a,b,c,d* is an *all different* workload while *a,a,a,a* is an *all same* workload. This effectively brackets the expected diversity in any workload – including server, parallel, and multithreaded workloads. Hence, we expect our results to be generalizable across a wide range of applications.

Structure	Area (mm^2)	Power (W)
8KB-DM cache	0.4	0.638
16KB-2-way cache	0.745	1.018
32KB-4-way cache	1.495	1.744
64KB-4-way cache	2.6	1.869
64KB-4-way dual-ported cache	5.05	3.932
64-entry ITLB	0.119	0.126
128-entry ITLB	0.238	0.186
16-entry InstQ (Int/FP)	0.063, 0.203, 0.721 ($IW = 1, 2, 4$)	0.129, 0.266, 0.565 ($IW = 1, 2, 4$)
32-entry InstQ (Int/FP)	0.086, 0.273, 0.991 ($IW = 1, 2, 4$)	0.144, 0.301, 0.655 ($IW = 1, 2, 4$)
64-entry InstQ	0.16, 0.505, 2.596 ($IW = 1, 2, 4$)	0.186, 0.394, 0.899 ($IW = 1, 2, 4$)
32-entry lsQ single-port (Int/FP)	0.1	0.161
32-entry lsQ dual-ported (Int/FP)	0.319	0.333
Branch Predictor	0.2	0.3
1 ALU	0.385	0.45
1 IntMul	0.295	0.45
1 FPU	0.728	0.9
32-entry Regfile	0.1, 0.339, 1.244 ($IW = 1, 2, 4$)	0.212, 0.439, 0.953 ($IW = 1, 2, 4$)
64-entry Regfile	0.137, 0.411, 1.5 ($IW = 1, 2, 4$)	0.367, 0.854, 1.897 ($IW = 1, 2, 4$)
128-entry Regfile	0.192, 0.611, 2.15 ($IW = 1, 2, 4$)	0.517, 1.154, 2.788 ($IW = 1, 2, 4$)
32-entry RAM Rename Table	0.049, 0.176, 0.668 ($IW = 1, 2, 4$)	0.137, 0.284, 0.606 ($IW = 1, 2, 4$)
32-entry ROB	0.04, 0.158, 0.533 ($IW = 1, 2, 4$)	0.07, 0.157, 0.311 ($IW = 1, 2, 4$)
64-entry ROB	0.06, 0.218, 0.753 ($IW = 1, 2, 4$)	0.1, 0.209, 0.451 ($IW = 1, 2, 4$)

Table 3: Derived Area and Power Estimates for Processor Components

Program	Description
ammp	Computational Chemistry
crafty	Game Playing:Chess
eon	Computer Visualization
mcf	Combinatorial Optimization
twolf	Place and Route Simulator
mgrid	Multi-grid Solver: 3D Potential Field
mesa	3-D Graphics Library
groff	Typesetting package
deltablue	Constraint Hierarchy Solver
adpcmc	Encoder for Adaptive Differential Pulse Code Modulation

Table 4: *Benchmarks used*

5.3.2 Evaluation methodology

As discussed before, there are over 2.2 billion distinct 4-core multiprocessors that can be constructed using our 480 distinct cores. We assume that the performance of a multiprocessor is the sum of the performance of each core of the multiprocessor, as described in Section 3. Each core is assumed to have a private L2 cache as well as a memory channel. This is the same architecture (private L2s) assumed in [8] and is supported by recent research comparing private and shared L2 caches for multi-core architectures [12]. We also validate that the results made with these assumptions still apply with shared L2 caches for our benchmarks (see the Appendix).

We find the single thread performance of each application on each core by simulating for 250 million cycles, after fast-forwarding an appropriate number of instructions [19]. This represents 4800 simulations. Simulations use a modified version of SMTSIM [22]. Scripts are used to calculate the performance of the multiprocessors using these single-thread performance numbers.

All results are presented for the best (oracular) static mapping of applications to cores. Note that realistic dynamic mapping can do better [11] – we show in Section 6 that dynamic mapping continues being useful for the best heterogeneous designs that our methodology produces. However, evaluating 2.2 billion multiprocessors becomes intractable if dynamic mapping is assumed.

5.3.3 Evaluation Metric

We use weighted speedup [21] for our evaluations. In this paper, weighted speedup measures the arithmetic sum of each running thread's IPC, divided by its IPC on the simplest core considered in this study when running alone. The IPC is derived by running a thread for a fixed amount of time. We believe that this metric guards against multiprocessor design points that produce artificial speedups by simply favoring high-IPC threads.

For completeness reasons, we also performed all our evaluations for total IPC as well and found that while the absolute results were different, there was no significant difference in trends or analysis.

6. ANALYSIS AND RESULTS

In the following sections, we examine the performance and resulting architecture of CMPs designed under various workload and design constraint assumptions. Section 6.1 considers a particular 4-threaded workload. Section 6.2 extends our analysis to a varied workload and a variety of different area and power constraints. In Section 6.3, we quantify the gains observed due to a methodology that allows non-monotonic cores on the processor. We examine the effect of different levels of thread-level parallelism for multiprogramming workloads in Section 6.4.

6.1 Analyzing multi-core processors for a given workload

This section examines a 4-core CMP design for a single, particular 4-thread workload. This serves a couple of purposes. First, it demonstrates the use of these design techniques for embedded MP systems where the workload is known, and also allows us to demonstrate (in a more concrete example) most of the design principles that are also true for the more general case (designing for an unknown permutation of applications). Thus, we consider here the four-threaded *all different* workload consisting of *eon, mesa, deltablue*, and *mcf*. These applications all have different characteristics (e.g. *deltablue* has high ILP while *mcf* is memory bound) and hence different execution requirements.

Figure 2 shows the highest performing multiprocessors for this workload for various number of core types. A processor with one

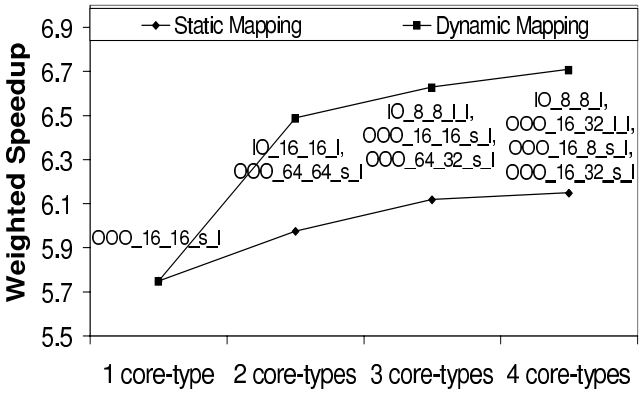

Figure 2: Core characteristics for the best performing CMPs for a given workload of eon, mesa, deltablue, and mcf. Area budget = 30mm^2. Power budget = 30W.

core type is a homogeneous CMP. A processor where all the 4 cores are different is a processor with 4 core types. We annotate the graph with descriptions of the actual cores chosen by our system. IO_icachesize_dcachesize_exec-units represents an inorder core. A core can have small (*exec_units=s*) or large (*exec_units=l*) number of functional units. OOO_icachesize_dcachesize_exec-units_physregs represents an out-of-order core. Such cores can have either small (*phys_regs=s*) or large (*phys_regs=l*) number of physical registers and ROB entries. The actual sizes that correspond to these settings appear in Table 1. (While not true in general, in this section no multi-issue cores are chosen, so the issue width does not appear in this notation). The best multiprocessors are arrived at assuming static mapping of applications to cores. However, results are also shown for dynamic mapping of applications to the cores of these multiprocessors – in this case, we use the same configuration of cores chosen assuming static mapping, but allow jobs to swap cores dynamically during execution, assuming a methodology similar to [11].

We find that the highest performing CMP indeed has all cores different where each core is well-suited for a particular application. While $IO\_8\_8\_l$ (in-order, 8K L1 caches, more functional units) is well-suited for *mesa*, $OOO\_16\_8\_s\_l$ (out-of-order, 16K Icache, 8 K Dcache, few functional units, large window) is well-suited for *mcf*. $OOO\_16\_32\_l\_l$ runs *deltablue* well and $OOO\_64\_32\_s\_l$ is tuned for *eon*. This processor has 7% higher throughput than the best homogeneous CMP design for static mapping. The best homogeneous CMP has an out-of-order core with 16K I and D caches, few functional units, and a large instruction window. This processor is a good fit only for *mcf*; it is an overfit for *mesa* and an underfit for *deltablue* and *eon* in terms of L1 cache and registers.

Even though these cores are highly specialized to particular applications, we get significant gains if we allow threads to move between cores over time. Thus, when dynamic mapping is assumed, benefits due to heterogeneity are even higher, as much as 16.7%, due to the ability to exploit intra-thread diversity. In the following sections, we are not able to evaluate dynamic mapping for all benchmark permutations, but this result confirms the expected result, that heterogeneous CMPs designed for static mapping only perform better when threads are allowed to move dynamically.

6.2 Analyzing multi-core processors for a given budget

This section extends our analysis in two ways. It considers a more general workload (designing for an unknown permutation of a set of applications) and a variety of area and power budgets. For every fixed area or power limit, an exhaustive search is performed to find the highest performing 4-core multiprocessor. For all budgets, the results shown assume that all contexts are busy. The configuration chosen is the one that gives the best average performance over all permutations of the applications.

Figure 3 shows the weighted speedup for the highest performing 4-core multiprocessors within an area budget of 40mm^2. The three lines correspond to different power budgets for the cores. The results are presented for two workload conditions – *all same*, when all the threads of a 4-threaded workload are the same and *all different*, when all the threads of a 4-threaded workload are different. These two conditions represent two extremes of heterogeneity. The points on the far left represent homogeneous CMP designs, all other points represent varying degrees of heterogeneity. Select points are labeled with a description of the core selection represented by that point, to aid in the following discussion.

The results lead to several interesting observations. First, we notice that the advantages of diversity exist even with the *all same* workload. This workload might represent parallel workloads with homogeneous threads, or perhaps a server handling requests with little diversity. Previous proposals discussed the advantages of heterogeneity only with heterogeneous workloads; however, we find that even homogeneous workloads achieve their best performance when at least one of the cores is well-suited for the application — a carefully constructed heterogeneous design ensures that whatever application is being used for the homogeneous runs, such a core likely exists. For example, for an area budget of 40mm^2 and a power budget of 30W, the best heterogeneous CMP for *all same* workloads outperforms the best homogeneous CMP by 4%.

Note that such a CMP is exploiting diversity *across* different homogeneous workloads even though there is no diversity *within* a workload (that is, we are finding a single best design for all of our *all same* workloads).

Second, we observe that the advantages due to heterogeneity for a fixed area budget depend largely on the power budget available — as shown by the shape of the lines corresponding to different power budgets. In this case (Figure 3), heterogeneity buys little additional performance with a generous power budget (50W), but is increasingly important as the budget becomes more tightly constrained. For example, in the *all-different* case, the best heterogeneous CMP outperforms the best homogeneous CMP by less than 1% when the power budget is 50W, by 8% when the power budget is 40W, and by 17% when the power budget is 30W. This can be explained by the the fact that without constraints, the homogeneous architecture can create "envelope" cores — cores that are over-provisioned for any single application, but able to run most applications with high performance. For example, for an area budget of 40mm^2, if the power budget is set high (50W), the "best" homogeneous architecture consists of 4 *OOO_64_64_l_l* cores (i.e., out-of-order, large caches, large window). This architecture is able to run both the memory-bound and processor-bound applications well. When the design is more constrained, we can only meet the needs of each application through heterogeneous designs that are customized to subsets of the applications.

We see these same trends in Figure 4, which shows results for four other area budgets. There is significant benefit to a diversity of cores as long as either area or power are reasonably constrained. For a power budget of 40W, a heterogeneous CMP outperforms the best homogeneous CMP by 8% when the area budget is 50mm^2 and by 10% when the budget is 30mm^2. A 11% improvement is possible for an area budget of 20mm^2 and a power budget of 30W.

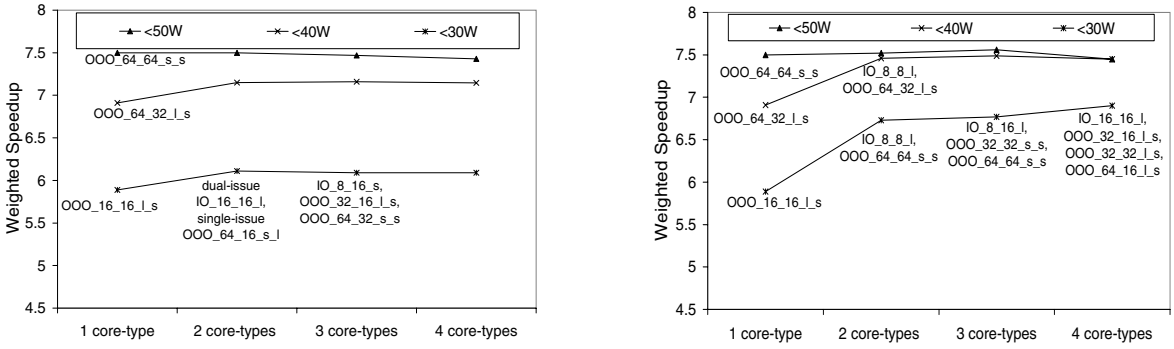

Figure 3: Throughput for *all-same* (left) and *all-different* (right) workloads, area budget=40mm^2

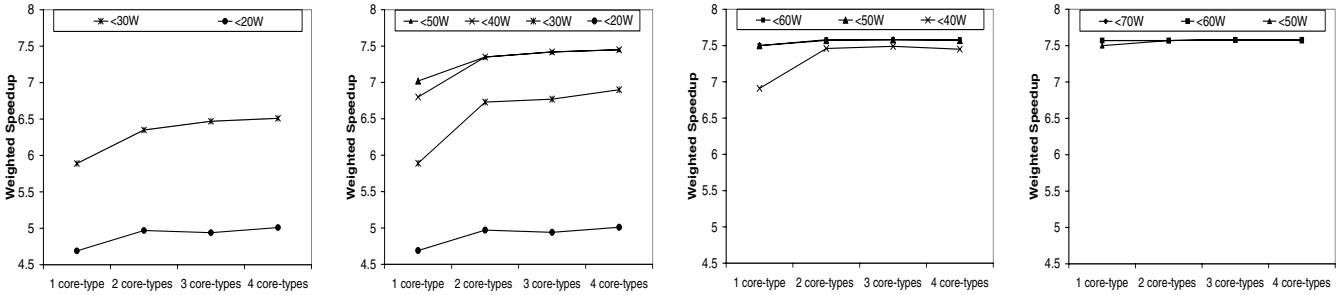

Figure 4: Throughput for *all-different* workloads for an area budget of (left to right) 20mm^2, 30mm^2, 50mm^2, and 60mm^2.

The power and area budgets also determine the amount of diversity needed for a multi-core architecture. In general, the more constrained the budget, the more benefits are accrued due to increased diversity. For example, considering the *all different* results in Figure 3, while having 4 core types results in the best performance when the power limit is 30W (17% improvement over the best homogeneous CMP), two core types (or in some cases, one) are sufficient to get more than 99% of the potential benefits for higher power limits. In some of the regions where moderate diversity is sufficient, two unique cores not only match configurations with higher diversity, but even beat them. In cases where higher diversity is optimal, the gains must still be compared against the design and test costs of more unique cores. For example, in the example above, the marginal performance of 4 core types over the best 2-type result is 3%, and may not justify the extra effort. Going from one core type to two core types, however, results in 14% performance improvement and presents a more compelling case.

Our results show that while having two core types is sufficient for getting most of the potential out of moderately power-limited designs, increased diversity results in significantly better performance for highly power-limited designs. These results underscore the increasing importance of single-ISA heterogeneous multi-core architectures for current and future processor designs. As designs become more aggressive, we will want to place more cores on the die (placing area pressure on the design), and power budgets per core will likely tighten even more severely.

Another way to interpret these results is that heterogeneous designs dampen the effects of constrained power budgets significantly. For example, in the 40mm^2 results, both homogeneous and heterogeneous solutions provide good performance with a 50W budget. However, the homogeneous design loses 9% performance with a 40W budget and 23% with a 30W budget. With a heterogeneous design, we can drop power to 40W with only a 2% penalty and down to 30W (a 40% power savings) with only a 9% performance loss.

Perhaps more illuminating than the raw performance of the best designs is what architectures actually provide the highest performance for a given area and power budget. We observe that there can be a significant difference between the cores of the best heterogeneous multiprocessor and the cores constituting the best homogeneous CMP. That is, the best heterogeneous multiprocessors cannot be constructed only by making slight modifications to the best homogeneous CMP design. Rather, they need to be designed from a clean slate. Consider, for example, the best multiprocessors for an area budget of 40mm^2 and a power budget of 40W. The best homogeneous CMP consists of single-issue *OOO_16_16_s_l* cores (out-of-order, 16K Icache, 16K Dcache, few functional units, large window). On the other hand, the best heterogeneous CMP with two types of cores, for *all different* workloads, consists of two single-issue in-order cores with 8KB L1 caches (*IO_8_8_s*) and two single-issue out-of-order cores with 64KB ICache, 32KB DCache and double the number of functional units (*OOO_64_32_1_s*).

A consistent observation also is the reliance on non-monotonicity. In several of our best heterogeneous configurations, we see that no core is a subset of any other core. For example, when the power budget is 30W, the best heterogeneous CMP for two core types for *all same* workloads consists of superscalar in-order cores (issue-width=2) and scalar out-of-order cores (issue-width=1), and outperforms the best homogeneous CMP by 4%. Even when all the cores are different, the best multiprocessor for *all different* workloads consists of a collection of one in-order core with 16KB L1 caches (*IO_16_16_s*), one out-of-order core with 32KB ICache and 16KB DCache (*OOO_32_16_s_s*), one in-order core with 32KB L1 caches (*IO_32_32_s*), and one out-of-order core with 64KB ICache and 16KB DCache (*OOO_64_16_s_s*). We explore this further in Section 6.3.

To summarize, the results show that the best heterogeneous CMP is not constructed of cores that make good general-purpose uniprocessor cores, or even those cores that would appear in good homogeneous multiprocessor architectures. Rather, the best way to

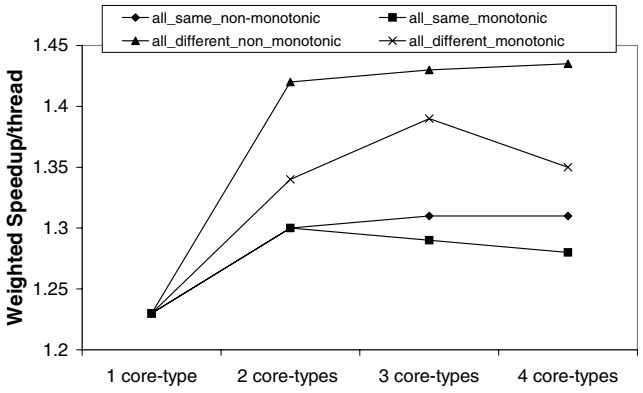

Figure 5: Benefits due to non-monotonicity of cores; area budget = 40mm^2, power budget = 30W

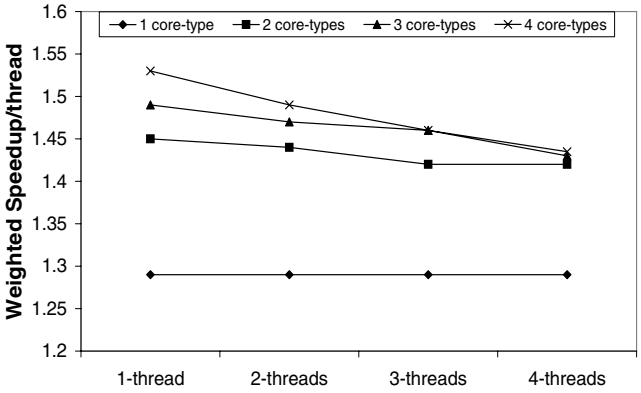

Figure 6: Performance of a 4-core processor designed for 4 threads when fewer threads are running

design a heterogeneous CMP is by tuning each individual core to a class of applications with common characteristics – we see this because the best designs typically contain cores poorly suited for some applications, but these designs will not have all cores poorly designed for a particular application. Such processors are advantageous even for completely homogeneous workloads and their benefits keep increasing as area and power budgets get tighter.

Note that all our results (even in the following sections) have been presented with various cutoff points (area/power budgets) for the ease of visualization. We analyzed the complete continuous data space, however, and also looked at finer intervals, to ensure that our conclusions were not particular to the cutoffs shown in these graphs.

6.3 Quantifying inefficiency due to monotonicity

Because all previous heterogeneous proposals either used different generations of the same family, or frequency scaling, they represent heterogeneous design methodologies that would fail to exploit non-monotonicity. Figure 5 compares the best monotonic designs against the best non-monotonic designs. We see that for the *all-same* workload, the benefits from non-monotonic configurations is small. However, with the heterogeneous workload, the non-monotonic designs outperform the monotonic much more significantly. The best non-monotonic design with 2 core types outperforms the best corresponding monotonic design by 7.5% and the best homogeneous CMP design by 15.4%. Also, while increasing diversity keeps increasing processor performance for non-monotonic designs, performance peaks at 3 core types for monotonic designs.

More generally (results not shown here), we find that the cost of monotonicity in terms of performance is greater when budgets are constrained. In fact, diversity beyond two core types has benefits for non-monotonic designs only for very constrained power and area budgets.

6.4 Varying Thread-Level Parallelism

While all our studies are done for 4-core processors, it is unrealistic to assume that the processor will always have at least four threads to run. Thus, we would like the same processor to run efficiently with thread-level parallelism (TLP) less than four.

Figure 6 shows the performance of the best derived 4-core processors ($area\ budget = 40mm^2,\ power\ budget = 30W$) when fewer threads are running. Thus, we optimize the cores for 4 threads, but run with fewer threads than the processor was designed for. As we can see, a heterogeneous processor designed for 4 threads continues to have benefits over the corresponding homogeneous processor even when fewer threads are run. A heterogeneous processor with 4 core types, for example, outperforms the best homogeneous 4-core design by 18.6% when there is only a single thread running.

For all levels of TLP, the gradation among the processors remain the same. In fact, the relative performance increases with fewer threads, even though the processor was designed assuming a TLP of 4. Because the four cores are customized to particular thread characteristics, threads are even better at finding the right core to run when there is less competition for cores. We observed similar results for other budgets as well.

We also investigate the effect of customizing a 4-core processor for a TLP different than the number of cores. Figure 7 shows the results. When designing for two threads or three threads, we actually assumed a mix of threading levels (1, 2, 3, or 4 threads) such that the average TLP was 2 or 3, respectively. Here we see that the benefits of heterogeneity are definitely more pronounced when the machine is designed for about the same threading level that is running, particularly with the *all-same* workload. In these particular experiments, we see nearly a 40% advantage from heterogeneity.

Of particular interest is the marked gain incurred by a single thread (TLP=1) when going from 2 to 3 unique core types. It would not have been too surprising if the best 1-thread heterogeneous processor had one big monolithic core which ran everything well, and a bunch of tiny cores never used. But instead, the best single-thread designs were more balanced and non-monotonic. This can be seen in these graphs, simply because having three cores types was better than two – in the case of the monolithic design, two would have been sufficient. Thus, we see here that we can get better *single-thread* performance with heterogeneous cores than with the largest monolithic core we are considering in this study.

6.5 Efficient Search Techniques

All the previous results were arrived at by evaluating every point in the search space – i.e. using exhaustive search. The reported best multiprocessors are indeed the best multiprocessors out of the ones considered. However, the full search considers over 2.2 billion distinct multiprocessor options. Then, each multiprocessor is evaluated on thousands of 4-thread workload permutations. As a result, search was slow. Additionally, this methodology is not scalable. In particular, the methodology becomes impractical when we increase the number of cores on die, the number of distinct core configurations considered, or the application set used to evaluate the architecture.

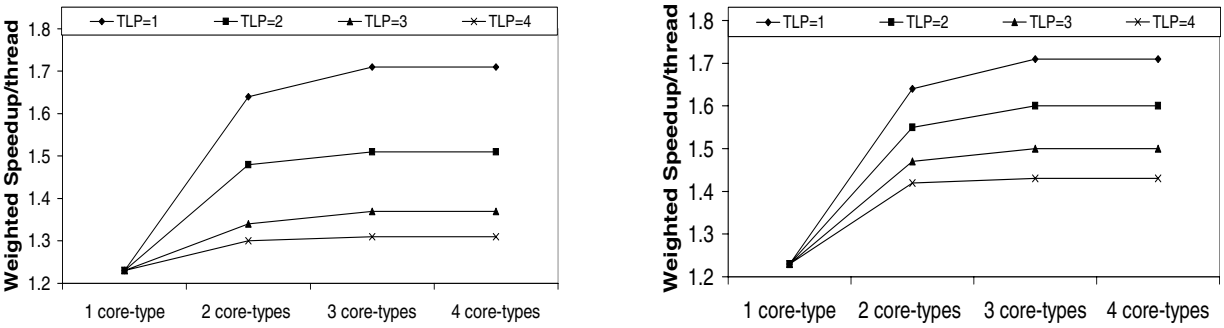

Figure 7: Throughput for *all-same* (left) and *all-different* (right) workloads when designed for different levels of thread level parallelism. In this figure, TLP indicates both the design target, and the number of threads running. Area budget=40mm^2, power budget=30W.

In those cases, a more efficient search algorithm is needed to navigate the design space. That is, instead of considering every multiprocessor design, one can evaluate only a subset of the space (e.g., a path through the space, seeking increasingly good designs). The choice of the subset determines the accuracy of evaluation.

As an example, we used a simple well-known search algorithm, *hill climbing* [18], to re-explore multiprocessors with an area budget of 40mm^2 and a power budget of 30W. The starting point was the four simplest cores, and we then made modifications to the multiprocessor, one at a time, such that the chosen modification resulted in the highest incremental performance per unit area out of all the incremental modifications possible. The modifications are made in small steps. Also, a change once made is never undone. This has the danger of the search getting stuck in local maxima, but results in much faster searches than *generalized hill climbing* [18] where the changes can be undone. The search stops when the available area budget or power budget is exceeded. We observed that the best heterogeneous multiprocessor yielded by *hill climbing* is 11% better that the best homogeneous CMP found using exhaustive search and is only 4.5% worse than the best heterogeneous CMP found using exhaustive search. Also, 86% fewer architectures were evaluated for this search. It is the subject of future work to experiment with other search algorithms, with which we hope to approach the effectiveness of exhaustive search more closely.

Note that search algorithms can be used even for fast design space exploration of uniprocessors or homogeneous CMPs using this same methodology.

7. CONCLUSIONS

This is the first attempt to look at the design of cores for a heterogeneous CMP. The goal is to determine how to do good heterogeneous CMP design for a given set of workloads and given area and power budgets. We try to identify the characteristics of the cores of a heterogeneous chip multiprocessor for the highest area or power efficiency, and quantify the benefits that can be obtained by doing a more custom design from the ground up. We also present a methodology for the design and optimization of the constituent processor cores given a set of target applications, as well as specific design budgets. We call these cores non-monotonic if their performance is not fully ordered over a range of different applications.

In contrast to previous work, we show that the best way to design a heterogeneous CMP is not to find individual cores that are well suited for the entire universe of applications, but rather to tune the cores to different classes of applications. We find that customizing cores to subsets of the workload results in processors that have greater performance and power benefits than previously proposed heterogeneous designs. An example such design outperformed the best homogeneous CMP design by 15.4% and the best fully customized monotonic design by 7.5%. There were performance and power improvements even for fully homogeneous workloads and for single-threaded workloads. Benefits are even greater when power and area budgets are increasingly constrained. Performance improvements of up to 40% are shown. Given current trends in processor design, we expect power to become increasingly constrained in the future, even in desktop and server markets. This means that the value of customizing core architecture as well as the benefits of aggressive heterogeneity will only increase with time.

Acknowledgments

The authors would like to thank Saisanthosh Balakrishnan, Jeff Collard, Soraya Ghiasi, Tomer Morad, and Partha Ranganathan for their valuable comments at various stages of this research, as well as the anonymous referees for their constructive suggestions. The research was funded in part by NSF grants CCF-0311683 and CCF-0541434, Semiconductor Research Corporation grant 2005-HJ-1313, and an IBM fellowship.

8. REFERENCES

[1] International Technology Roadmap for Semiconductors 2003, http://public.itrs.net.

[2] M. Annavaram, E. Grochowski, and J. Shen. Mitigating Amdahl's Law Through EPI Throttling. In *Proceedings of International Symposium on Computer Architecture*, 2005.

[3] S. Balakrishnan, R. Rajwar, M. Upton, and K. Lai. The impact of performance asymmetry in emerging multicore architectures. In *Proceedings of International Symposium on Computer Architecture*, 2005.

[4] S. Ghiasi and D. Grunwald. Aide de camp: Asymmetric dual core design for power and energy reduction. In *University of Colorado Technical Report CU-CS-964-03*, 2003.

[5] S. Ghiasi, T. Keller, and F. Rawson. Scheduling for heterogeneous processors in server systems. In *Proceedings of Computing Frontiers*, 2005.

[6] E. Grochowski, R. Ronen, J. Shen, and H. Wang. Best of both latency and throughput. In *Proceedings of IEEE International Conference on Computer Design*, 2004.

[7] S. Gupta, S. Keckler, and D. Burger. Technology independent area and delay estimates for microprocessor

building blocks. In *University of Texas at Austin Technical Report TR-00-05*, 1998.

[8] J. Huh, S. W. Keckler, and D. Burger. Exploring the design space of future CMPs. In *Proceedings of the International Conference on Parallel Architectures and Compilation Techniques*, 2001.

[9] R. Kotla, A. Devgan, S. Ghiasi, T. Keller, and F. Rawson. Characterizing the impact of different memory-intensity levels. In *Proceedings of IEEE 7th Annual Workshop on Workload Characterization*, 2004.

[10] R. Kumar, K. I. Farkas, N. P. Jouppi, P. Ranganathan, and D. M. Tullsen. Single-ISA Heterogeneous Multi-core Architectures: The Potential for Processor Power Reduction. In *International Symposium on Microarchitecture*, Dec. 2003.

[11] R. Kumar, D. M. Tullsen, P. Ranganathan, N. P. Jouppi, and K. I. Farkas. Single-ISA Heterogeneous Multi-core Architectures for Multithreaded Workload Performance. In *International Symposium on Computer Architecture*, June 2004.

[12] R. Kumar, V. Zyuban, and D. M. Tullsen. Interconnections in multi-core architectures: Understanding mechanisms, overheads and scaling. In *Proceedings of International Symposium on Computer Architecture*, 2005.

[13] J. Li and J. Martinez. Power-performance implications of thread-level parallelism in chip multiprocessors. In *Proceedings of International Symposium on Performance Analysis of Systems and Software*, 2005.

[14] S. McFarling. Combining branch predictors. Technical Report TN-36, DEC-WRL, June 1993.

[15] T. Morad, U. Weiser, and A. Kolodny. ACCMP - assymetric cluster chip-multiprocessing. In *CCIT Technical Report 488*, 2004.

[16] T. Y. Morad, U. C. Weiser, A. Kolodny, M. Valero, and E. Ayguade. Performance, power efficiency and scalability of asymmetric cluster chip multiprocessors. In *Computer Architecture Letters, Vol 4*, July 2005.

[17] J. M. Mulder, N. T. Quach, and M. J. Flynn. An area model for on-chip memories and its applications. In *IEEE Journal of Solid State Circuits, Vol 26, No. 2*, Feb. 1991.

[18] E. Rich and K. Knight. *Artificial Intelligence, 2nd Edition*. Morgan Kaufmann, 1991.

[19] T. Sherwood, E. Perelman, G. Hamerly, and B. Calder. Automatically characterizing large scale program behavior. In *Tenth International Comference on Architectural Support for Programming Languages and Operating Systems*, Oct. 2002.

[20] P. Shivakumar and N. Jouppi. CACTI 3.0: An integrated cache timing, power and area model. In *Technical Report 2001/2, Compaq Computer Corporation*, Aug. 2001.

[21] A. Snavely and D. Tullsen. Symbiotic jobscheduling for a simultaneous multithreading architecture. In *Eighth International Conference on Architectural Support for Programming Languages and Operating Systems*, Nov. 2000.

[22] D. Tullsen. Simulation and modeling of a simultaneous multithreading processor. In *22nd Annual Computer Measurement Group Conference*, Dec. 1996.

APPENDIX

Our results assume that the performance of a multiprocessor is simply the sum of the single-thread performance of the individual cores. Our confidence in this methodology stems from the fact that each core on the multiprocessor gets a private 1MB 4-way L2 cache and a private memory channel, that we are using multi-programmed, not parallel, workloads, and that the SPEC benchmarks (and others used in the study) do not generate heavy off-chip traffic. In this section, we validate the accuracy of this assumption with full multithreaded simulation of particular points, accounting for interactions beyond the L2 cache.

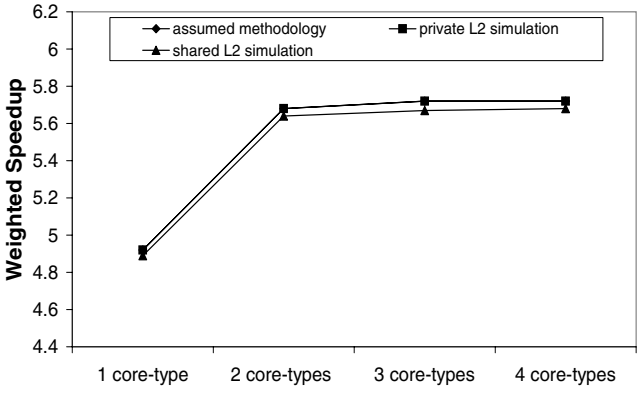

Figure 8: Comparing the results using assumed methodology against full simulation results: area budget = 40mm^2, power budget = 30W

Less obvious is whether these results, and the methodology we follow, applies to the case where L2 caches are shared and interactions between cores are greater. We cannot fully validate all results without full multithreaded simulation of all configurations. Thus, we reduce the full validation to two questions.

First, is the performance of the configurations found to be "best" accurate, particular with respect to the relative performance of homogeneous and heterogeneous designs? Second, is the performance ordering of configurations competing for the designation of "best" configuration preserved with more accurate simulation?

To answer the first question, we consider the highest performing multiprocessors for various number of core types for an area budget of 40mm^2 and a power budget of 30W. We performed full simulation for the four multiprocessors for various 4-threaded *all different* workloads. We performed simulations assuming that each core has a private 1MB 4-way L2 cache. We also performed simulations assuming that all cores shared a 4MB, 4-banked 4-way L2 cache. Figure 8 shows the results.

As can be seen from the graph, the relative ordering of the four multiprocessors remains the same. In fact, the simulations assuming private L2 caches lies on top of the line drawn using the methodology assumed in the paper. Even when the L2 cache is shared, the average performance differs by no more than 2%.

For checking the second condition, we chose 5 multiprocessors with two core-types whose performance, as determined by the assumed methodology, fell within the top 50 percentile. The multiprocessors are chosen such that they are equally spread through that range (one from the 90-100th percentile, one from the 80-90th, etc.). Then we performed full simulation for those multiprocessors and compared them against the performance of the multiprocessors using the assumed methodology. We found that the relative ordering of these multiprocessors in terms of performance remains the same in all three cases (performance using assumed methodology, full simulation assuming private L2 caches and full simulation assuming a shared L2 cache). We considered other datapoints as well and observed no significant difference in the trends or analysis.